# Recommendations_with_IBM

June 24, 2021

## 1 Recommendations with IBM

In this notebook, you will be putting your recommendation skills to use on real data from the IBM Watson Studio platform.

You may either submit your notebook through the workspace here, or you may work from your local machine and submit through the next page. Either way assure that your code passes the project RUBRIC. **Please save regularly.**

By following the table of contents, you will build out a number of different methods for making recommendations that can be used for different situations.

### 1.1 Table of Contents

I. Section **??** II. Section **??** III. Section **??** IV. Section **??** V. Section **??** VI. Section **??**

At the end of the notebook, you will find directions for how to submit your work. Let's get started by importing the necessary libraries and reading in the data.

```
In [55]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import project_tests as t
         import pickle

         %matplotlib inline

         df = pd.read_csv('data/user-item-interactions.csv')
         df_content = pd.read_csv('data/articles_community.csv')
         del df['Unnamed: 0']
         del df_content['Unnamed: 0']

         # Show df to get an idea of the data
         df.head()
```

```
Out[55]:    article_id                                              title  \
         0      1430.0  using pixiedust for fast, flexible, and easier...
         1      1314.0       healthcare python streaming application demo
         2      1429.0         use deep learning for image classification
         3      1338.0          ml optimization using cognitive assistant
         4      1276.0          deploy your python model as a restful api
```

```
                                              email
0  ef5f11f77ba020cd36e1105a00ab868bbdbf7fe7
1  083cbdfa93c8444beaa4c5f5e0f5f9198e4f9e0b
2  b96a4f2e92d8572034b1e9b28f9ac673765cd074
3  06485706b34a5c9bf2a0ecdac41daf7e7654ceb7
4  f01220c46fc92c6e6b161b1849de11faacd7ccb2
```

In [56]:
```
# Show df_content to get an idea of the data
df_content.head(10)
```

Out[56]:

```
                                            doc_body  \
0  Skip navigation Sign in SearchLoading...\r\n\r...
1  No Free Hunch Navigation * kaggle.com\r\n\r\n ...
2   * Login\r\n * Sign Up\r\n\r\n * Learning Pat...
3  DATALAYER: HIGH THROUGHPUT, LOW LATENCY AT SCA...
4  Skip navigation Sign in SearchLoading...\r\n\r...
5  Compose is all about immediacy. You want a new...
6  UPGRADING YOUR POSTGRESQL TO 9.5Share on Twitt...
7  Follow Sign in / Sign up 135 8 * Share\r\n * 1...
8  * Host\r\n * Competitions\r\n * Datasets\r\n *...
9  THE GRADIENT FLOW\r\nDATA / TECHNOLOGY / CULTU...

                                       doc_description  \
0  Detect bad readings in real time using Python ...
1  See the forest, see the trees. Here lies the c...
2  Heres this weeks news in Data Science and Bi...
3  Learn how distributed DBs solve the problem of...
4  This video demonstrates the power of IBM DataS...
5          Using Compose's PostgreSQL data browser.
6  Upgrading your PostgreSQL deployment to versio...
7  For a company like Slack that strives to be as...
8  Kaggle is your home for data science. Learn ne...
9  [A version of this post appears on the OReill...

                                    doc_full_name doc_status  article_id
0  Detect Malfunctioning IoT Sensors with Streami...       Live           0
1  Communicating data science: A guide to present...       Live           1
2         This Week in Data Science (April 18, 2017)       Live           2
3  DataLayer Conference: Boost the performance of...       Live           3
4      Analyze NY Restaurant data using Spark in DSX       Live           4
5              Browsing PostgreSQL Data with Compose       Live           5
6                   Upgrading your PostgreSQL to 9.5       Live           6
7                            Data Wrangling at Slack       Live           7
8                             Data Science Bowl 2017       Live           8
9  Using Apache Spark to predict attack vectors a...       Live           9
```

In [57]:
```
# 17 Articles don't have a email, hence not user interaction
```

```
df.isnull().sum(axis=0).to_frame()
df_content.isnull().sum(axis=0).to_frame()
```

Out[57]:

| | |
|---|---|
| | 0 |
| doc_body | 14 |
| doc_description | 3 |
| doc_full_name | 0 |
| doc_status | 0 |
| article_id | 0 |

### 1.1.1 Part I : Exploratory Data Analysis

Use the dictionary and cells below to provide some insight into the descriptive statistics of the data.

1. What is the distribution of how many articles a user interacts with in the dataset? Provide a visual and descriptive statistics to assist with giving a look at the number of times each user interacts with an article.

In [58]:
```
#df_content.shape
print(df.shape)
#unique users
n_users = df['email'].nunique()
print('unique users:',n_users)
```

```
(45993, 3)
unique users: 5148
```

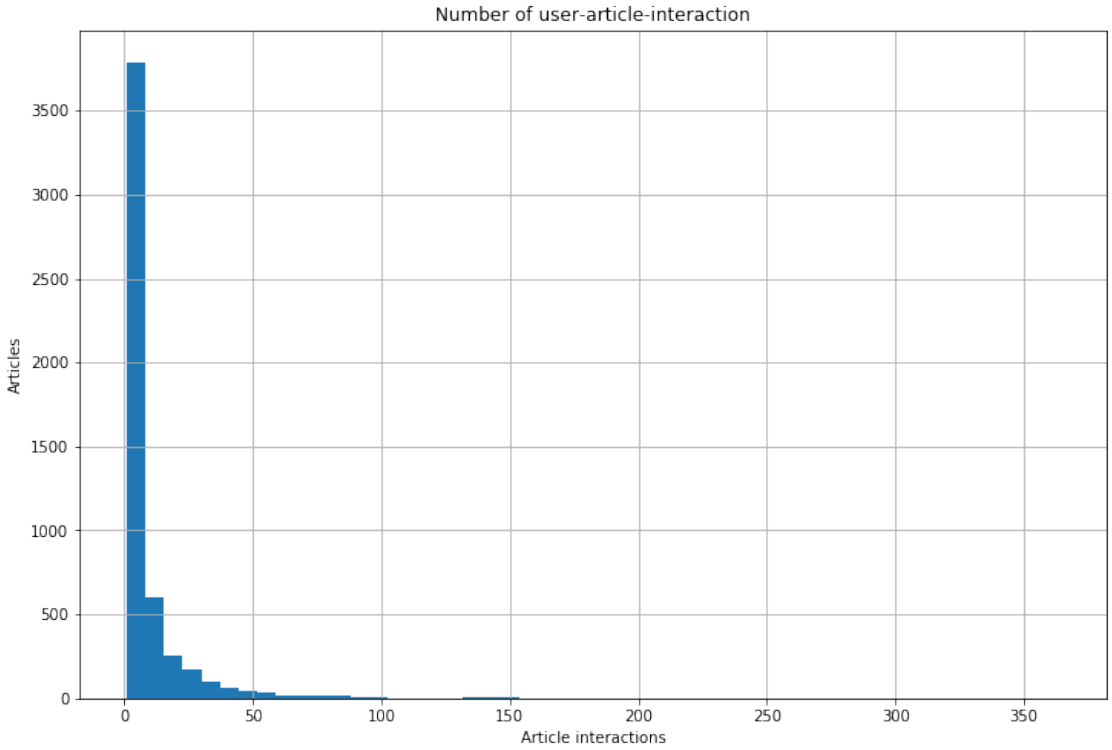
In [59]:
```
hist_df = df.groupby('email')['article_id'].count().to_frame().reset_index()
hist_df.head(5)
```

Out[59]:

| | email | article_id |
|---|---|---|
| 0 | 0000b6387a0366322d7fbfc6434af145adf7fed1 | 13 |
| 1 | 001055fc0bb67f71e8fa17002342b256a30254cd | 4 |
| 2 | 00148e4911c7e04eeff8def7bbbdaf1c59c2c621 | 3 |
| 3 | 001a852ecbd6cc12ab77a785efa137b2646505fe | 6 |
| 4 | 001fc95b90da5c3cb12c501d201a915e4f093290 | 2 |

In [60]:
```
# hist_df.hist(column='article_id', bins=30, grid=True, figsize=(12,8))

hist = df.groupby('email')['article_id'].count().hist(bins=50, grid=True, figsize=(12,8
plt.title('Number of user-article-interaction')
plt.xlabel('Article interactions')
plt.ylabel('Articles')
```

Out[60]:
```
Text(0,0.5,'Articles')
```




### 1.1.2 Execute to answer question

In [61]:
```
# Fill in the median and maximum number of user_article interactios below

# 50% of individuals interact with ____ number of articles or fewer.
df_grouped = df.groupby('email')['article_id'].count().to_frame()
median_val = df_grouped['article_id'].median(axis = 0)
print('median', median_val)


# The maximum number of user-article interactions by any 1 user is ______.
max_views_by_user = df.groupby('email')['article_id'].count().to_frame().sort_values('a
print('max_views_by_user', max_views_by_user)
```

```
median 3.0
max_views_by_user 364
```

2. Explore and remove duplicate articles from the **df_content** dataframe.

In [62]:
```
# Find and explore duplicate articles
df_content[df_content.duplicated(subset=['article_id'])==True]
```

```
Out[62]:                                                  doc_body  \
         365  Follow Sign in / Sign up Home About Insight Da...
         692  Homepage Follow Sign in / Sign up Homepage * H...
         761  Homepage Follow Sign in Get started Homepage *...
         970  This video shows you how to construct queries ...
         971  Homepage Follow Sign in Get started * Home\r\n...

                                              doc_description  \
         365  During the seven-week Insight Data Engineering...
         692  One of the earliest documented catalogs was co...
         761  Todays world of data science leverages data f...
         970  This video shows you how to construct queries ...
         971  If you are like most data scientists, you are ...

                                           doc_full_name doc_status  article_id
         365                      Graph-based machine learning       Live          50
         692  How smart catalogs can turn the big data flood...       Live         221
         761  Using Apache Spark as a parallel processing fr...       Live         398
         970                             Use the Primary Index       Live         577
         971  Self-service data preparation with IBM Data Re...       Live         232
```

### 1.1.3 Drop duplicates in df_content

```
In [63]: # Remove any rows that have the same article_id - only keep the first
         df_content.drop_duplicates(subset=['article_id'],inplace=True)
         df_content['article_id'].count()
```

```
Out[63]: 1051
```

3. Use the cells below to find:

**a.** The number of unique articles that have an interaction with a user.
**b.** The number of unique articles in the dataset (whether they have any interactions or not). **c.** The number of unique users in the dataset. (excluding null values) **d.** The number of user-article interactions in the dataset.

### 1.1.4 Execute to answer Questions

```
In [64]: # The number of unique articles that have at least one interaction
         unique_articles = df[df['email'].notnull()].nunique()[0]
         print('a. unique_articles_interaction:', unique_articles)

         # The number of unique articles on the IBM platform
         total_articles = df_content['article_id'].nunique()
         print('b. total_articles:', total_articles)

         # The number of unique users
         # Notice you may also find the number of unique users as 5149 if you count the null use
         # However, this is hard to catch without mapping first!
```

```
unique_users = df['email'].nunique()
print('c. unique_users:', unique_users)

#The number of user-article interactions
user_article_interactions = df.shape[0]
#df[df['email'].notnull()].count()[0]
print('d. user_article_interactions:', user_article_interactions)
```

```
a. unique_articles_interaction: 714
b. total_articles: 1051
c. unique_users: 5148
d. user_article_interactions: 45993
```

4. Use the cells below to find the most viewed **article_id**, as well as how often it was viewed. After talking to the company leaders, the `email_mapper` function was deemed a reasonable way to map users to ids. There were a small number of null values, and it was found that all of these null values likely belonged to a single user (which is how they are stored using the function below).

```
In [65]: # The most viewed article in the dataset as a string with one value following the decim
         most_viewed_article_id = str(df.groupby('article_id')['email'].count().sort_values(asce
         print('e. most_viewed_article_id',most_viewed_article_id)

         # The most viewed article in the dataset was viewed how many times?
         max_views = df.groupby('article_id')['email'].count().sort_values(ascending=False).iloc
         print('f. max_views',max_views)
```

```
e. most_viewed_article_id 1429.0
f. max_views 937
```

```
In [66]: ## No need to change the code here - this will be helpful for later parts of the notebo
         # Run this cell to map the user email to a user_id column and remove the email column

         def email_mapper():
             coded_dict = dict()
             cter = 1
             email_encoded = []

             for val in df['email']:
                 if val not in coded_dict:
                     coded_dict[val] = cter
                     cter+=1

                 email_encoded.append(coded_dict[val])
             return email_encoded

         email_encoded = email_mapper()
         del df['email']
```

```
df['user_id'] = email_encoded

# show header
df.head()
```

Out[66]:

| | article_id | title | user_id |
|---|---|---|---|
| 0 | 1430.0 | using pixiedust for fast, flexible, and easier... | 1 |
| 1 | 1314.0 | healthcare python streaming application demo | 2 |
| 2 | 1429.0 | use deep learning for image classification | 3 |
| 3 | 1338.0 | ml optimization using cognitive assistant | 4 |
| 4 | 1276.0 | deploy your python model as a restful api | 5 |

```
In [67]: ## If you stored all your results in the variable names above,
         ## you shouldn't need to change anything in this cell

         sol_1_dict = {
             '`50% of individuals have _____ or fewer interactions.`': median_val,
             '`The total number of user-article interactions in the dataset is ______.`': user_a
             '`The maximum number of user-article interactions by any 1 user is ______.`': max_v
             '`The most viewed article in the dataset was viewed _____ times.`': max_views,
             '`The article_id of the most viewed article is ______.`': most_viewed_article_id,
             '`The number of unique articles that have at least 1 rating ______.`': unique_artic
             '`The number of unique users in the dataset is ______`': unique_users,
             '`The number of unique articles on the IBM platform`': total_articles
         }

         # Test your dictionary against the solution
         t.sol_1_test(sol_1_dict)
```

```
It looks like you have everything right here! Nice job!
```

### 1.1.5 Part II: Rank-Based Recommendations

Unlike in the earlier lessons, we don't actually have ratings for whether a user liked an article or not. We only know that a user has interacted with an article. In these cases, the popularity of an article can really only be based on how often an article was interacted with.

1. Fill in the function below to return the **n** top articles ordered with most interactions as the top. Test your function using the tests below.

```
In [68]: def get_top_articles(n, df=df):
             '''
             INPUT:
             n - (int) the number of top articles to return
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             top_articles - (list) A list of the top 'n' article titles
```

```
    '''
    # Your code here
    top_articles = df.groupby(['article_id','title'])['user_id'].count().sort_values(as

    return list(top_articles['title'])


   # return top_articles # Return the top article titles from df (not df_content)
    pass

def get_top_article_ids(n, df=df):
    '''
    INPUT:
    n - (int) the number of top articles to return
    df - (pandas dataframe) df as defined at the top of the notebook

    OUTPUT:
    top_articles - (list) A list of the top 'n' article titles

    '''
    # Your code here
    #change data type for test assert
    tmp = df
    tmp['article_id'] = tmp['article_id'].astype(str)
    top_articles = list(tmp.groupby('article_id')['user_id'].count().sort_values(ascend

    return top_articles # Return the top article ids
```

In [69]:
```
#top_articles = list(x['title'])

print(get_top_articles(10))
print(get_top_article_ids(10))
```

```
['use deep learning for image classification', 'insights from new york car accident reports', 'v
['1429.0', '1330.0', '1431.0', '1427.0', '1364.0', '1314.0', '1293.0', '1170.0', '1162.0', '1304
```

In [70]:
```
# Test your function by returning the top 5, 10, and 20 articles
top_5 = get_top_articles(5)
top_10 = get_top_articles(10)
top_20 = get_top_articles(20)

# Test each of your three lists from above
t.sol_2_test(get_top_articles)
```

```
Your top_5 looks like the solution list! Nice job.
Your top_10 looks like the solution list! Nice job.
Your top_20 looks like the solution list! Nice job.
```

### 1.1.6 Part III: User-User Based Collaborative Filtering

1. Use the function below to reformat the **df** dataframe to be shaped with users as the rows and articles as the columns.

- Each **user** should only appear in each **row** once.
- Each **article** should only show up in one **column**.
- **If a user has interacted with an article, then place a 1 where the user-row meets for that article-column**. It does not matter how many times a user has interacted with the article, all entries where a user has interacted with an article should be a 1.
- **If a user has not interacted with an item, then place a zero where the user-row meets for that article-column**.

Use the tests to make sure the basic structure of your matrix matches what is expected by the solution.

```
In [ ]: df.head()
```

```
In [71]: # create the user-article matrix with 1's and 0's
         # Manual approach
         def create_user_item_matrix_(df):
             '''
             INPUT:
             df - pandas dataframe with article_id, title, user_id columns

             OUTPUT:
             user_item - user item matrix

             Description:
             Return a matrix with user ids as rows and article ids on the columns with 1 values
             an article and a 0 otherwise
             '''
             tmp = df.sort_values('article_id',ascending=True)

             # Create an empty dataframe
             column = list(tmp['article_id'].unique()) # .drop('title',axis=1)
             user_item = pd.DataFrame(columns=column, index=range(tmp['user_id'].max()+1))

             for i,j in df.iterrows():

                 user_id = j[[2][0]]
                 article_id = j[[0][0]]

                 user_item.iloc[user_id][article_id] = 1

             user_item = user_item.fillna(0)
             #drop first row
```

```
    user_item = user_item.iloc[1: , :]

    return user_item# return the user_item matrix


# Approach with unstack or pivot
def create_user_item_matrix(df):
    '''
    INPUT:
    df - pandas dataframe with article_id, title, user_id columns

    OUTPUT:
    user_item - user item matrix

    Description:
    Return a matrix with user ids as rows and article ids on the columns with 1 values
    an article and a 0 otherwise
    '''
    # Fill in the function here
    # replace title
    user_item = df.drop('title',axis=1)

    user_item.drop_duplicates(keep='last',inplace=True)

    # add column interact with 1 hence it show a user has interacted with an article
    user_item['interact'] = 1

    # unstack matrix and replace nans with 0
    #user_item = user_item.groupby(['user_id', 'article_id'])['interact'].max().unstack

    user_item = user_item.pivot(index='user_id', columns='article_id')['interact'].fill
    return user_item # return the user_item matrix

user_item = create_user_item_matrix(df)
print('finish')
```

```
finish
```

```
In [72]: user_item.head(2)
```

```
Out[72]: article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
         user_id
         1           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         2           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0

         article_id  1016.0  ...    977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
         user_id             ...
```

```
1              0.0  ...      0.0   0.0    1.0    0.0    0.0    0.0    0.0
2              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0

article_id  993.0  996.0  997.0
user_id
1             0.0    0.0    0.0
2             0.0    0.0    0.0

[2 rows x 714 columns]
```

```
In [73]: ## Tests: You should just need to run this cell.  Don't change the code.
         assert user_item.shape[0] == 5149, "Oops!  The number of users in the user-article matr
         assert user_item.shape[1] == 714, "Oops!  The number of articles in the user-article ma
         assert user_item.sum(axis=1)[1] == 36, "Oops!  The number of articles seen by user 1 do
         print("You have passed our quick tests!  Please proceed!")
```

```
You have passed our quick tests!  Please proceed!
```

2. Complete the function below which should take a user_id and provide an ordered list of the most similar users to that user (from most similar to least similar). The returned result should not contain the provided user_id, as we know that each user is similar to him/herself. Because the results for each user here are binary, it (perhaps) makes sense to compute similarity as the dot product of two users.

Use the tests to test your function.

```
In [ ]: #similarity = user_item[user_item.index == 20].dot(user_item.T)
        #similarity
```

```
In [74]: # Approach with np array -> buggy leads to errors
         def find_similar_users_(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user_id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

             OUTPUT:
             similar_users - (list) an ordered list where the closest users (largest dot product
                             are listed first

             Description:
             Computes the similarity of every pair of users based on the dot product
             Returns an ordered

             '''
             #create numpy array
             user_item_np = np.array(user_item)
```

```
    # compute similarity of each user to the provided user
    user_idx = np.where(user_item.index == user_id)[0][0]

    #create dot product
    dot_product = user_item_np.dot(np.transpose(user_item_np))

    # find the most similar user_ids
    most_similar_users = np.where(dot_product[user_idx] == np.max(dot_product[user_idx]


    return most_similar_users # return a list of the users in order from most to least

# approach directly on dataframe
def find_similar_users(user_id, user_item=user_item):
    '''
    INPUT:
    user_id - (int) a user_id
    user_item - (pandas dataframe) matrix of users by articles:
                1's when a user has interacted with an article, 0 otherwise

    OUTPUT:
    similar_users - (list) an ordered list where the closest users (largest dot product
                    are listed first

    Description:
    Computes the similarity of every pair of users based on the dot product
    Returns an ordered

    '''

    # https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dot.h

    # compute similarity of each user to the provided user
    similarity = user_item[user_item.index == user_id].dot(user_item.T)

    # create list of just the ids and sort
    most_similar = similarity.sort_values(user_id, axis=1, ascending=False)

    # get user_id
    most_similar_users = most_similar.columns.tolist()

    # remove the own user's id
    most_similar_users.pop(0)



    return most_similar_users # return a list of the users in order from most to least
```

```
In [75]: # Do a spot check of your function
         print("The 10 most similar users to user 1 are: {}".format(find_similar_users(1)[:10]))
         print("The 5 most similar users to user 3933 are: {}".format(find_similar_users(3933)[:
         print("The 3 most similar users to user 46 are: {}".format(find_similar_users(46)[:3]))
```

```
The 10 most similar users to user 1 are: [3933, 23, 3782, 203, 4459, 3870, 131, 4201, 46, 5041]
The 5 most similar users to user 3933 are: [3933, 23, 3782, 203, 4459]
The 3 most similar users to user 46 are: [4201, 3782, 23]
```

3. Now that you have a function that provides the most similar users to each user, you will want to use these users to find articles you can recommend. Complete the functions below to return the articles you would recommend to each user.

```
In [76]: def get_article_names(article_ids, df=df):
             '''
             INPUT:
             article_ids - (list) a list of article ids
             df - (pandas dataframe) df as defined at the top of the notebook

             OUTPUT:
             article_names - (list) a list of article names associated with the list of article
                             (this is identified by the title column)
             '''
             # Your code here
             article_names = df.drop_duplicates(subset = ['article_id']).sort_values('article_id
             article_names = article_names[article_names['article_id'].isin(article_ids)]

             return list(article_names['title']) # Return the article names associated with list


         def get_user_articles(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

             OUTPUT:
             article_ids - (list) a list of the article ids seen by the user
             article_names - (list) a list of article names associated with the list of article
                             (this is identified by the doc_full_name column in df_content)

             Description:
             Provides a list of the article_ids and article titles that have been seen by a user
             '''
             # Your code here
```

```
    seen = list(user_item.iloc[user_id-1])
    cols = list(user_item.columns)

    article_ids = []

    for idx, (see, col) in enumerate(zip(seen,cols)):
        if see > 0:
            article_ids.append(col)


    article_names = get_article_names(article_ids)

    return article_ids, article_names # return the ids and names


def user_user_recs(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user

    Description:
    Loops through the users based on closeness to the input user_id
    For each user - finds articles the user hasn't seen before and provides them as rec
    Does this until m recommendations are found

    Notes:
    Users who are the same closeness are chosen arbitrarily as the 'next' user

    For the user where the number of recommended articles starts below m
    and ends exceeding m, the last items are chosen arbitrarily

    '''
    # Your code here

    # empty list of recommondations
    recs = []

    # number of added articles
    num_added = 0

    # find similar users: return id list
    similar_users = find_similar_users(user_id)

    # get list of already seen articles
```

```
            article_ids_tmp, article_names_tmp = get_user_articles(user_id)

            for sim_user_id in similar_users:
                #print(sim_user_id)
                article_ids, article_names = get_user_articles(sim_user_id)

                article_names = list(set(article_names)-set(article_names_tmp))
                #print(article_names)
                if num_added  < m:
                    for name in article_names:
                        #print(name)
                        recs.append(name)
                        num_added += 1

                        if num_added >= m:
                            break # end loop
                else:
                    break # end loop

            return recs # return your recommendations for this user_id
```

In [77]:
```
# Check Results
user_user_recs(3933, 10) # Return 10 recommendations for user 1


#get_user_articles(2)[0]
#user_user_recs(20, m=10)
#user_item
```

Out[77]:
```
['airbnb data for analytics: vancouver listings',
 'a tensorflow regression model to predict house values',
 'this week in data science (april 25, 2017)',
 'simple graphing with ipython and\xa0pandas',
 'use sql with data in hadoop python',
 'visualize data with the matplotlib library',
 'spark 2.1 and job monitoring available in dsx',
 'graph-based machine learning',
 'times world university ranking analysis',
 'deploy your python model as a restful api']
```

In [78]:
```
# Test your functions here - No need to change this code - just run this cell
assert set(get_article_names(['1024.0', '1176.0', '1305.0', '1314.0', '1422.0', '1427.0
assert set(get_article_names(['1320.0', '232.0', '844.0'])) == set(['housing (2015): un
assert set(get_user_articles(20)[0]) == set(['232.0', '844.0', '1320.0'])
assert set(get_user_articles(20)[1]) == set(['housing (2015): united states demographic
assert set(get_user_articles(2)[0]) == set(['1024.0', '1176.0', '1305.0', '1314.0', '14
assert set(get_user_articles(2)[1]) == set(['using deep learning to reconstruct high-re
print("If this is all you see, you passed all of our tests!  Nice job!")
```

```
If this is all you see, you passed all of our tests!  Nice job!
```

4. Now we are going to improve the consistency of the **user_user_recs** function from above.

- Instead of arbitrarily choosing when we obtain users who are all the same closeness to a given user - choose the users that have the most total article interactions before choosing those with fewer article interactions.
- Instead of arbitrarily choosing articles from the user where the number of recommended articles starts below m and ends exceeding m, choose articles with the articles with the most total interactions before choosing those with fewer total interactions. This ranking should be what would be obtained from the **top_articles** function you wrote earlier.

```
In [79]: def find_similar_users_values(user_id, user_item=user_item):
             '''
             INPUT:
             user_id - (int) a user_id
             user_item - (pandas dataframe) matrix of users by articles:
                         1's when a user has interacted with an article, 0 otherwise

             OUTPUT:
             similar_users - (list) an ordered list where the closest users (largest dot product
                             are listed first

             Description:
             Computes the similarity of every pair of users based on the dot product
             Returns an ordered

             '''
             # https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dot.h

             # compute similarity of each user to the provided user
             similarity = user_item[user_item.index == user_id].dot(user_item.T)

             # create list of just the ids and sort
             most_similar = similarity.sort_values(user_id, axis=1, ascending=False)

             # get similarity value
             most_similar_values = list(most_similar.values[0])

             # get user_id
             most_similar_users = most_similar.columns.tolist()

             #drop actual user id
             most_similar_users.pop(0)
             most_similar_values.pop(0)
             # remove the own user's id
```

```
    return most_similar_users, most_similar_values  # return a list of the users in ord


def get_top_sorted_users(user_id, df=df, user_item=user_item):
    '''
    INPUT:
    user_id - (int)
    df - (pandas dataframe) df as defined at the top of the notebook
    user_item - (pandas dataframe) matrix of users by articles:
            1's when a user has interacted with an article, 0 otherwise


    OUTPUT:
    neighbors_df - (pandas dataframe) a dataframe with:
                    neighbor_id - is a neighbor user_id
                    similarity - measure of the similarity of each user to the provided
                    num_interactions - the number of articles viewed by the user - if a

    Other Details - sort the neighbors_df by the similarity and then by number of inter
                    highest of each is higher in the dataframe

    '''
    # Your code here
    # get user_ids and similarity values
    most_similar_users, most_similar_values = find_similar_users_values(user_id)

    most_viewed = df.groupby('user_id')['article_id'].count().to_frame()
    most_viewed = most_viewed[most_viewed.index.isin(most_similar_users)].reset_index()
    most_viewed = list(most_viewed['article_id']) #get list

    # Create dataframe
    neighbors_df = pd.DataFrame({'neighbor_id':most_similar_users,\
                                 'similarity':most_similar_values,\
                                 'num_interactions':most_viewed}).set_index('neighbor_i

    # Return the dataframe specified in the doc_string
    return neighbors_df.sort_values(['similarity','num_interactions'],ascending=False)


def user_user_recs_part2(user_id, m=10):
    '''
    INPUT:
    user_id - (int) a user id
    m - (int) the number of recommendations you want for the user

    OUTPUT:
    recs - (list) a list of recommendations for the user by article id
    rec_names - (list) a list of recommendations for the user by article title
```

```
Description:
Loops through the users based on closeness to the input user_id
For each user - finds articles the user hasn't seen before and provides them as rec
Does this until m recommendations are found

Notes:
* Choose the users that have the most total article interactions
before choosing those with fewer article interactions.

* Choose articles with the articles with the most total interactions
before choosing those with fewer total interactions.

'''
# Your code here

# empty list of recommondations
recs = []
rec_names = []

# number of added articles
num_added = 0

# get list of already seen articles
article_ids_tmp, article_names_tmp = get_user_articles(user_id)

# Loop through sorted users
for sort_user_id in get_top_sorted_users(20).index:

    article_ids, article_names = get_user_articles(sort_user_id)

    # remove alreday seen articles from user
    article_names = list(set(article_names)-set(article_names_tmp))
    article_ids = list(set(article_ids)-set(article_ids_tmp))

    #print(article_names)
    if num_added  < m:
        for id, name in zip(article_ids,article_names):
            #print(name)
            recs.append(id)
            rec_names.append(name)
            num_added += 1

            if num_added >= m:
                break # end loop
    else:
        break # end loop
```

```
            return recs, rec_names
```

```
In [80]: # Quick spot check - don't change this code - just use it to test your functions
         rec_ids, rec_names = user_user_recs_part2(20, 10)
         print("The top 10 recommendations for user 20 are the following article ids:")
         print(rec_ids)
         print()
         print("The top 10 recommendations for user 20 are the following article names:")
         print(rec_names)
```

```
The top 10 recommendations for user 20 are the following article ids:
['1271.0', '1403.0', '1402.0', '1328.0', '1410.0', '1280.0', '1154.0', '1304.0', '681.0', '1444.

The top 10 recommendations for user 20 are the following article names:
['education (2015): united states demographic measures', 'uci: adult - predict income', 'income
```

5. Use your functions from above to correctly fill in the solutions to the dictionary below. Then test your dictionary against the solution. Provide the code you need to answer each following the comments below.

```
In [81]: ### Tests with a dictionary of results
         user1_most_sim =  get_top_sorted_users(1).index[0] # Find the user that is most similar
         print('user1_most_sim:',user1_most_sim)
         user131_10th_sim = get_top_sorted_users(131).index[9]# Find the 10th most similar user
         print('user131_10th_sim:',user131_10th_sim)
```

```
user1_most_sim: 3933
user131_10th_sim: 242
```

```
In [82]: ## Dictionary Test Here
         sol_5_dict = {
             'The user that is most similar to user 1.': user1_most_sim,
             'The user that is the 10th most similar to user 131': user131_10th_sim,
         }

         t.sol_5_test(sol_5_dict)
```

```
This all looks good!  Nice job!
```

6. If we were given a new user, which of the above functions would you be able to use to make recommendations? Explain. Can you think of a better way we might make recommendations? Use the cell below to explain a better method for new users.

This case would describe a cold start problem. Thus, since we do not have any information about newly introduced users, we can not make any recommendations with SVD function. Instead, we can use content based or knowledge based recommendations.

7. Using your existing functions, provide the top 10 recommended articles you would provide for the a new user below. You can test your function against our thoughts to make sure we are all on the same page with how we might make a recommendation.

```
In [83]: new_user = '0.0'

         # What would your recommendations be for this new user '0.0'?  As a new user, they have
         # Provide a list of the top 10 article ids you would give to

         # since there is no similarity available, i would simply provide the ten most interacte
         # Your recommendations here
         new_user_recs = get_top_article_ids(10)
         new_user_recs

Out[83]: ['1429.0',
          '1330.0',
          '1431.0',
          '1427.0',
          '1364.0',
          '1314.0',
          '1293.0',
          '1170.0',
          '1162.0',
          '1304.0']

In [84]: assert set(new_user_recs) == set(['1314.0','1429.0','1293.0','1427.0','1162.0','1364.0'

         print("That's right!  Nice job!")

That's right!  Nice job!
```

### 1.1.7 Part IV: Content Based Recommendations (EXTRA - NOT REQUIRED)

Another method we might use to make recommendations is to perform a ranking of the highest ranked articles associated with some term. You might consider content to be the **doc_body**, **doc_description**, or **doc_full_name**. There isn't one way to create a content based recommendation, especially considering that each of these columns hold content related information.

1. Use the function body below to create a content based recommender. Since there isn't one right answer for this recommendation tactic, no test functions are provided. Feel free to change the function inputs if you decide you want to try a method that requires more input values. The input values are currently set with one idea in mind that you may use to make content based recommendations. One additional idea is that you might want to choose the most popular recommendations that meet your 'content criteria', but again, there is a lot of flexibility in how you might make these recommendations.

### 1.1.8 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: def make_content_recs():
            '''
            INPUT:
```

```
    OUTPUT:

    '''
```

2. Now that you have put together your content-based recommendation system, use the cell below to write a summary explaining how your content based recommender works. Do you see any possible improvements that could be made to your function? Is there anything novel about your content based recommender?

#### 1.1.9 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

**Write an explanation of your content based recommendation system here.**

3. Use your content-recommendation system to make recommendations for the below scenarios based on the comments. Again no tests are provided here, because there isn't one right answer that could be used to find these content based recommendations.

#### 1.1.10 This part is NOT REQUIRED to pass this project. However, you may choose to take this on as an extra way to show off your skills.

```
In [ ]: # make recommendations for a brand new user


        # make a recommendations for a user who only has interacted with article id '1427.0'
```

#### 1.1.11 Part V: Matrix Factorization

In this part of the notebook, you will build use matrix factorization to make article recommendations to the users on the IBM Watson Studio platform.

1. You should have already created a **user_item** matrix above in **question 1** of **Part III** above. This first question here will just require that you run the cells to get things set up for the rest of **Part V** of the notebook.

```
In [85]: # Load the matrix here
         user_item_matrix = pd.read_pickle('user_item_matrix.p')

In [86]: # quick look at the matrix
         user_item_matrix.head()

Out[86]: article_id  0.0  100.0  1000.0  1004.0  1006.0  1008.0  101.0  1014.0  1015.0  \
         user_id
         1           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         2           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         3           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         4           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0
         5           0.0    0.0     0.0     0.0     0.0     0.0    0.0     0.0     0.0

         article_id  1016.0  ...    977.0  98.0  981.0  984.0  985.0  986.0  990.0  \
         user_id             ...
```

```
1              0.0  ...      0.0   0.0    1.0    0.0    0.0    0.0    0.0
2              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0
3              0.0  ...      1.0   0.0    0.0    0.0    0.0    0.0    0.0
4              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0
5              0.0  ...      0.0   0.0    0.0    0.0    0.0    0.0    0.0

article_id  993.0  996.0  997.0
user_id
1             0.0    0.0    0.0
2             0.0    0.0    0.0
3             0.0    0.0    0.0
4             0.0    0.0    0.0
5             0.0    0.0    0.0

[5 rows x 714 columns]
```

2. In this situation, you can use Singular Value Decomposition from numpy on the user-item matrix. Use the cell to perform SVD, and explain why this is different than in the lesson.

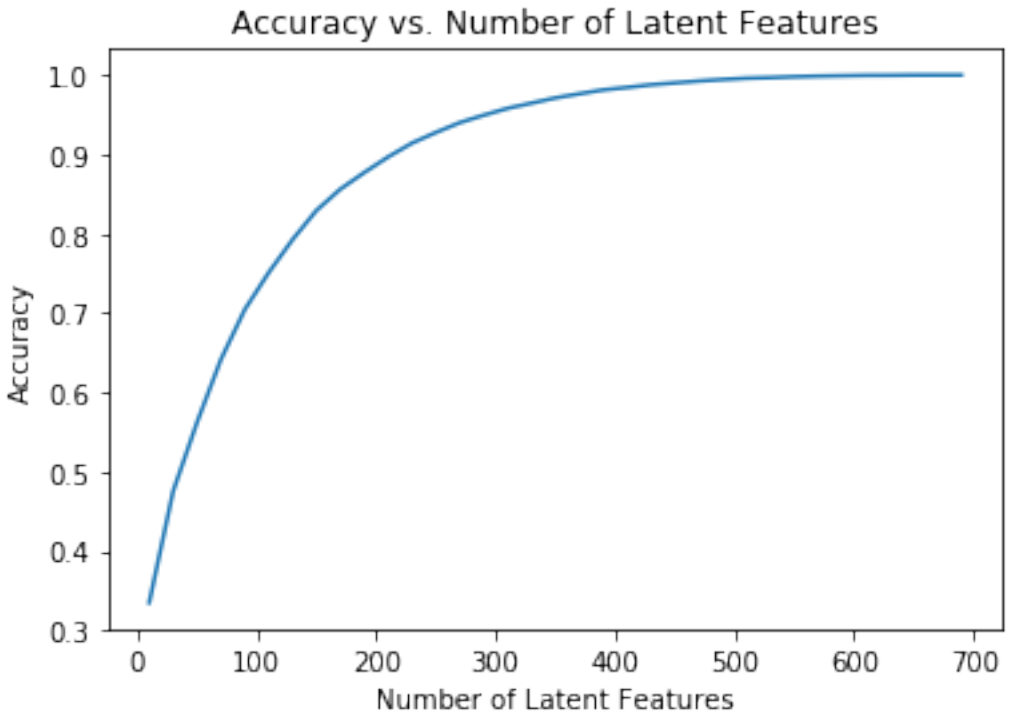
In [87]:
```
# Perform SVD on the User-Item Matrix Here

#create numpy array
user_item_np = np.array(user_item)

# use the built in to get the three matrices
u, s, vt = np.linalg.svd(user_item_np, full_matrices=True)
```

**Provide your response here.**

3. Now for the tricky part, how do we choose the number of latent features to use? Running the below cell, you can see that as the number of latent features increases, we obtain a lower error rate on making predictions for the 1 and 0 values in the user-item matrix. Run the cell below to get an idea of how the accuracy improves as we increase the number of latent features.

In [88]:
```
num_latent_feats = np.arange(10,700+10,20)
sum_errs = []

for k in num_latent_feats:
    # restructure with k latent features
    s_new, u_new, vt_new = np.diag(s[:k]), u[:, :k], vt[:k, :]

    # take dot product
    user_item_est = np.around(np.dot(np.dot(u_new, s_new), vt_new))

    # compute error for each prediction to actual value
    diffs = np.subtract(user_item_matrix, user_item_est)

    # total errors and keep track of them
    err = np.sum(np.sum(np.abs(diffs)))
    sum_errs.append(err)
```

```
plt.plot(num_latent_feats, 1 - np.array(sum_errs)/df.shape[0]);
plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.title('Accuracy vs. Number of Latent Features');
```




4. From the above, we can't really be sure how many features to use, because simply having a better way to predict the 1's and 0's of the matrix doesn't exactly give us an indication of if we are able to make good recommendations. Instead, we might split our dataset into a training and test set of data, as shown in the cell below.

Use the code from question 3 to understand the impact on accuracy of the training and test sets of data with different numbers of latent features. Using the split below:

- How many users can we make predictions for in the test set?
- How many users are we not able to make predictions for because of the cold start problem?
- How many articles can we make predictions for in the test set?
- How many articles are we not able to make predictions for because of the cold start problem?

```
In [104]: df_train = df.head(40000)
          df_test = df.tail(5993)

          print('unique train users:',df_train['user_id'].nunique())
```

```
print('unique test users:',df_test['user_id'].nunique())

def create_test_and_train_user_item(df_train, df_test):
    '''
    INPUT:
    df_train - training dataframe
    df_test - test dataframe

    OUTPUT:
    user_item_train - a user-item matrix of the training dataframe
                      (unique users for each row and unique articles for each column)
    user_item_test - a user-item matrix of the testing dataframe
                    (unique users for each row and unique articles for each column)
    test_idx - all of the test user ids
    test_arts - all of the test article ids

    '''

    # Your code here

    user_item_train = create_user_item_matrix_(df_train)

    user_item_test = create_user_item_matrix_(df_test)

    test_idx = list(user_item_test.index)

    test_arts = list(user_item_test.columns)


    return user_item_train, user_item_test, test_idx, test_arts

user_item_train, user_item_test, test_idx, test_arts = create_test_and_train_user_item
```

```
unique train users: 4487
unique test users: 682
```

```
In [90]: print(len(test_arts))

         #len(test_idx)
```

```
574
```

```
In [111]: #How many users can we make predictions for in the test set?
          n_pred = len(set(df_train['user_id'].unique()).intersection(set(df_test['user_id'].uni
          print('user predictions:',n_pred)
```

```
        # How many users in the test set are we not able to make predictions for because of th
        cs_users = len(test_idx)-len(set(user_item_train.index).intersection(set(test_idx)))
        print('coldstart users:',cs_users)

        #How many articles can we make predictions for in the test set?
        n_arct = len(set(user_item_train.columns).intersection(set(test_arts)))
        print('number articles:',n_arct)

        #How many articles in the test set are we not able to make predictions for because of
        cs_cols = len(set(user_item_train.columns).intersection(set(test_arts))) - len(test_ar
        print('cold start articles:',cs_cols)

        # Coldstart Problem: All elements in index or column which are not part of the trainin
        # but in test set, can't be used in SVD
```

```
user predictions: 20
coldstart users: 662
number articles: 574
cold start articles: 0
```

In [108]:
```
        # Replace the values in the dictionary below
        a = 662
        b = 574
        c = 20  # since there are 4487 user in train and 5149 in test, why 20????
        d = 0


        # sol_4_dict = {
        #     'How many users can we make predictions for in the test set?': len(num_predictio
        #     'How many users in the test set are we not able to make predictions for because
        #     ': len(test_arts), # letter here,
        #     'How many articles in the test set are we not able to make predictions for becau
        #     'How many movies can we make predictions for in the test set?': 0
        # }

        #t.sol_4_test(sol_4_dict)


        #### Seeems like that something is from with the sol_4_test!!!!!

        sol_3_dict = {
            'How many movies can we make predictions for in the test set?': n_arct, #b: 574
            'How many movies in the test set are we not able to make predictions for because c
            'How many users can we make predictions for in the test set?': n_pred, #c: 20
            'How many users in the test set are we not able to make predictions for because of
        }

        t.sol_4_test(sol_3_dict)
```

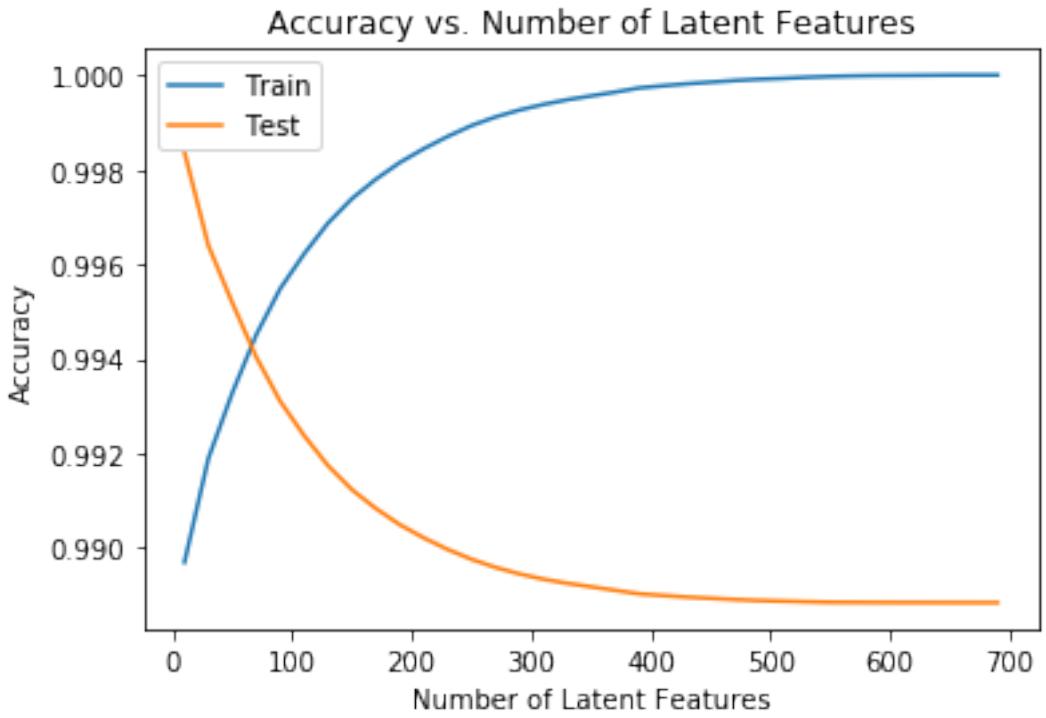
```
Awesome job!  That's right!  All of the test movies are in the training data, but there are only
```

5. Now use the **user_item_train** dataset from above to find U, S, and V transpose using SVD. Then find the subset of rows in the **user_item_test** dataset that you can predict using this matrix decomposition with different numbers of latent features to see how many features makes sense to keep based on the accuracy on the test data. This will require combining what was done in questions 2 - 4.

Use the cells below to explore how well SVD works towards making predictions for recommendations on the test data.

```
In [109]: #create numpy array
          user_item_train_np = np.array(user_item_train)

          # fit SVD on the user_item_train matrix
          u_train, s_train, vt_train = np.linalg.svd(user_item_train_np)
          # fit svd similar to above then use the cells below

In [110]: # Use these cells to see how well you can use the training
          # decomposition to predict on test data


          # get subset of user index and columns
          test_user_idx = user_item_train.index.isin(test_idx)
          test_art_idx = user_item_train.columns.isin(test_arts)

          u_test = u_train[test_user_idx, :]
          vt_test = vt_train[:, test_art_idx]

In [95]: # find the users in train and test data
         user_train_test = np.intersect1d(user_item_test.index, user_item_train.index)
         user_item_test_pred = user_item_test[user_item_test.index.isin(user_train_test)]

In [96]: num_latent_feats = np.arange(10,700+10,20)
         sum_errs_train = []
         sum_errs_test = []

         for k in num_latent_feats:
             # restructure with k latent features
             s_train_l, u_train_l, vt_train_l = np.diag(s_train[:k]), u_train[:, :k], vt_train[:
             u_test_l, vt_test_l = u_test[:, :k], vt_test[:k, :]

             # take dot product
             user_item_train_est = np.around(np.dot(np.dot(u_train_l, s_train_l), vt_train_l))
             pred = np.around(np.dot(np.dot(u_test_l, s_train_l), vt_test_l))


             # compute error for each prediction to actual value
             diffs_train = np.subtract(user_item_train, user_item_train_est)
```

```
    diffs_test = np.subtract(user_item_test_pred, pred)

    # total errors and keep track of them
    err_train = np.sum(np.sum(np.abs(diffs_train)))
    err_test = np.sum(np.sum(np.abs(diffs_test)))
    sum_errs_train.append(err_train)
    sum_errs_test.append(err_test)


plt.plot(num_latent_feats, 1- np.array(sum_errs_train)/(user_item_train.shape[0]
                                                        * user_item_test_pred.shape[1]
plt.plot(num_latent_feats, 1- np.array(sum_errs_test)/(user_item_test_pred.shape[0]
                                                       * user_item_test_pred.shape[1])

plt.xlabel('Number of Latent Features');
plt.ylabel('Accuracy');
plt.legend();
plt.title('Accuracy vs. Number of Latent Features');
```




6. Use the cell below to comment on the results you found in the previous question. Given the circumstances of your results, discuss what you might do to determine if the recommendations you make with any of the above recommendation systems are an improvement to how users currently find articles?

**Answers** First of all, because of the given test and train dataset, there is only a number of 20 users given in both. Thus, only predictions of 20 users can be given. However we have a total number of unique train users of 4487 and a total number of unique test users of 682. Because of that, another strategy of splitting the datasets could lead to more matches.

We can denote from the graph, that by the increasing number of latent features, the accuracy is tending to 1. On the other hand, the accuarcy of the testset decrease by the number of the latent features, hence there is an indication of overfitting in this model. Because of that, i would suggest to reduce the number latent feature to improve the accuracy.

But in general, since the accuracy is of the test set is still around 0.99, the results arent that bad.

A general problem of the entire recommendation is, that we do not know, if the users actually liked the articels or not. By assuming it because of the number of interactions can lead to false results. To make better recommendations, the dataset should have information about the articles rating.

### Extras Using your workbook, you could now save your recommendations for each user, develop a class to make new predictions and update your results, and make a flask app to deploy your results. These tasks are beyond what is required for this project. However, from what you learned in the lessons, you certainly capable of taking these tasks on to improve upon your work here!

## 1.2 Conclusion

Congratulations! You have reached the end of the Recommendations with IBM project!

**Tip**: Once you are satisfied with your work here, check over your report to make sure that it is satisfies all the areas of the rubric. You should also probably remove all of the "Tips" like this one so that the presentation is as polished as possible.

## 1.3 Directions to Submit

Before you submit your project, you need to create a .html or .pdf version of this notebook in the workspace here. To do that, run the code cell below. If it worked correctly, you should get a return code of 0, and you should see the generated .html file in the workspace directory (click on the orange Jupyter icon in the upper left).

Alternatively, you can download this report as .html via the **File** > **Download as** submenu, and then manually upload it into the workspace directory by clicking on the orange Jupyter icon in the upper left, then using the Upload button.

Once you've done this, you can submit your project by clicking on the "Submit Project" button in the lower right here. This will create and submit a zip file with this .ipynb doc and the .html or .pdf version you created. Congratulations!

```
In [ ]: from subprocess import call
        call(['python', '-m', 'nbconvert', 'Recommendations_with_IBM.ipynb'])
```

```
In [ ]:
```

```
In [ ]: # Backup stuff
```

```
 # find similar users: return id list
similar_users = find_similar_users(20)

# get list of already seen articles
article_ids_tmp, article_names_tmp = get_user_articles(20)

total_articles = set()

for sim_user_id in similar_users:

    article_ids, article_names = get_user_articles(sim_user_id)

    total_articles.update(article_ids)

total_articles = list(total_articles-set(article_ids_tmp))
top_articles = list(get_top_article_ids(len(total_articles)))
top_articles = sorted(set(top_articles) & set(total_articles), key = top_articles.index)
top_articles

total_articles
```